# Twitter Sentimental Analysis

# AUTHOR: MOHANKRISHNA

## Importing Libraries

In [3]:
```
# Importing necessary libraries
import re
import pickle
import numpy as np
import pandas as pd

import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Setting a more appealing style for plots
sns.set(style='whitegrid')
plt.style.use('ggplot')  # Use ggplot style for better visual appeal
```

## Importing Dataset

In [5]:
```
# Importing the dataset and processing it
DATASET_COLUMNS  = ["sentiment", "ids", "date", "flag", "user", "text"]
DATASET_ENCODING = "ISO-8859-1"

# Reading the dataset and selecting only necessary columns
dataset = pd.read_csv('twitter_sentiment_data.csv', encoding=DATASET_ENCODING,

# Renaming the columns to 'sentiment' and 'text'
dataset.columns = ['sentiment', 'text']

# Replacing sentiment values for easier interpretation (0 = negative, 1 = posi
dataset['sentiment'] = dataset['sentiment'].replace(4, 1)

# Dropping any missing values for cleanliness
dataset.dropna(inplace=True)

# Displaying the first few rows of the dataset for inspection
print(dataset.head())
```

```
   sentiment                                               text
0          0  is upset that he can't update his Facebook by ...
1          0  @Kenichan I dived many times for the ball. Man...
2          0    my whole body feels itchy and like its on fire
3          0  @nationwideclass no, it's not behaving at all....
4          0                      @Kwesidei not the whole crew
```

**Text Preprocessing** is traditionally an important step for Natural Language Processing (NLP) tasks. It transforms text into a more digestible form so that machine learning algorithms can perform better.

The Preprocessing steps taken are:

1. Lower Casing: Each text is converted to lowercase. Replacing URLs: Links starting with "http" or "https" or "www" are replaced by "URL".
2. Replacing Emojis: Replace emojis by using a pre-defined dictionary containing emojis along with their meaning. (eg: ":)" to "EMOJIsmile")
3. Replacing Usernames: Replace @Usernames with word "USER". (eg: "@Kaggle" to "USER")
4. Removing Non-Alphabets: Replacing characters except Digits and Alphabets with a space.
5. Removing Consecutive letters: 3 or more consecutive letters are replaced by 2 letters. (eg: "Heyyyy" to "Heyy")
6. Removing Short Words: Words with length less than 2 are removed.
7. Removing Stopwords: Stopwords are the English words which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. (eg: "the", "he", "have")
8. Lemmatizing: Lemmatization is the process of converting a word to its base form. (e.g: “Great” to “Good”)

In [7]:
```
# Defining dictionary containing all emojis with their meanings.
emojis = {':)': 'smile', ':-)': 'smile', ';d': 'wink', ':-E': 'vampire', ':(':
          ':-(': 'sad', ':-<': 'sad', ':P': 'raspberry', ':O': 'surprised',
          ':-@': 'shocked', ':@': 'shocked',':-$': 'confused', ':\\': 'annoyed
          ':#': 'mute', ':X': 'mute', ':^)': 'smile', ':-&': 'confused', '$_$'
          '@@': 'eyeroll', ':-!': 'confused', ':-D': 'smile', ':-0': 'yell', '
          '<(-_-)>': 'robot', 'd[-_-]b': 'dj', ":'-)": 'sadsmile', ';)': 'wink
          ';-)': 'wink', 'O:-)': 'angel','O*-)': 'angel','(:-D': 'gossip', '=^

## Defining set containing all stopwords in english.
stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all', 'am', '
             'and','any','are', 'as', 'at', 'be', 'because', 'been', 'before',
             'being', 'below', 'between','both', 'by', 'can', 'd', 'did', 'do'
             'does', 'doing', 'down', 'during', 'each','few', 'for', 'from',
             'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here',
             'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in
             'into','is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma',
             'me', 'more', 'most','my', 'myself', 'now', 'o', 'of', 'on', 'onc
             'only', 'or', 'other', 'our', 'ours','ourselves', 'out', 'own', '
             's', 'same', 'she', "shes", 'should', "shouldve",'so', 'some', 's
             't', 'than', 'that', "thatll", 'the', 'their', 'theirs', 'them',
             'themselves', 'then', 'there', 'these', 'they', 'this', 'those',
             'through', 'to', 'too','under', 'until', 'up', 've', 'very', 'was
             'we', 'were', 'what', 'when', 'where','which','while', 'who', 'wh
             'why', 'will', 'with', 'won', 'y', 'you', "youd","youll", "youre"
             "youve", 'your', 'yours', 'yourself', 'yourselves']
```

In [8]:
```
from nltk.stem import WordNetLemmatizer
```

In [9]:
```
def preprocess(textdata):
    processedText = []

    # Create Lemmatizer and Stemmer
    wordLemm = WordNetLemmatizer()

    # Regex patterns
    urlPattern = r"((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*)"
    userPattern       = '@[^\s]+'
    alphaPattern      = "[^a-zA-Z0-9]"
    sequencePattern   = r"(.)\1\1+"
    seqReplacePattern = r"\1\1"

    for tweet in textdata:
        tweet = tweet.lower()

        # Replace all URls with 'URL'
        tweet = re.sub(urlPattern,' URL',tweet)
        # Replace all emojis.
        for emoji in emojis.keys():
            tweet = tweet.replace(emoji, "EMOJI" + emojis[emoji])
        # Replace @USERNAME to 'USER'.
        tweet = re.sub(userPattern,' USER', tweet)
        # Replace all non alphabets.
        tweet = re.sub(alphaPattern, " ", tweet)
        # Replace 3 or more consecutive letters by 2 letter.
        tweet = re.sub(sequencePattern, seqReplacePattern, tweet)
        tweetwords = ''
        for word in tweet.split():
            # Checking if the word is a stopword.
            # If word not in stopwordlist
            if len(word)>1:

                word = wordLemm.lemmatize(word)

                tweetwords += (word+' ')
        processedText.append(tweetwords)

    return processedText
```

In [10]:
```
# Assuming 'dataset' has already been loaded and contains the 'text' column

import time

# Function for text preprocessing (example function, adjust as needed)
def preprocess(texts):
    # Example preprocessing: converting to lowercase and removing special char
    return [re.sub(r'\W', ' ', str(text).lower()) for text in texts]

# Record the start time
t = time.time()

# Preprocess the 'text' column from the dataset
processed_text = preprocess(dataset['text'])

print(f'Text Preprocessing complete.')
print(f'Time Taken: {round(time.time()-t)} seconds')

# Display the first few rows of the processed text
print(processed_text[:5])
```

```
Text Preprocessing complete.
Time Taken: 17 seconds
['is upset that he can t update his facebook by texting it    and might cry a
s a result  school today also  blah ', ' kenichan i dived many times for the
ball  managed to save 50   the rest go out of bounds', 'my whole body feels i
tchy and like its on fire ', ' nationwideclass no  it s not behaving at all
i m mad  why am i here  because i can t see you all over there  ', ' kwesidei
not the whole crew ']
```

# Analysing the data

Now we're going to analyse the preprocessed data to get an understanding of it. We'll plot **Word Clouds** for **Positive and Negative** tweets from our dataset and see which words occur the most.

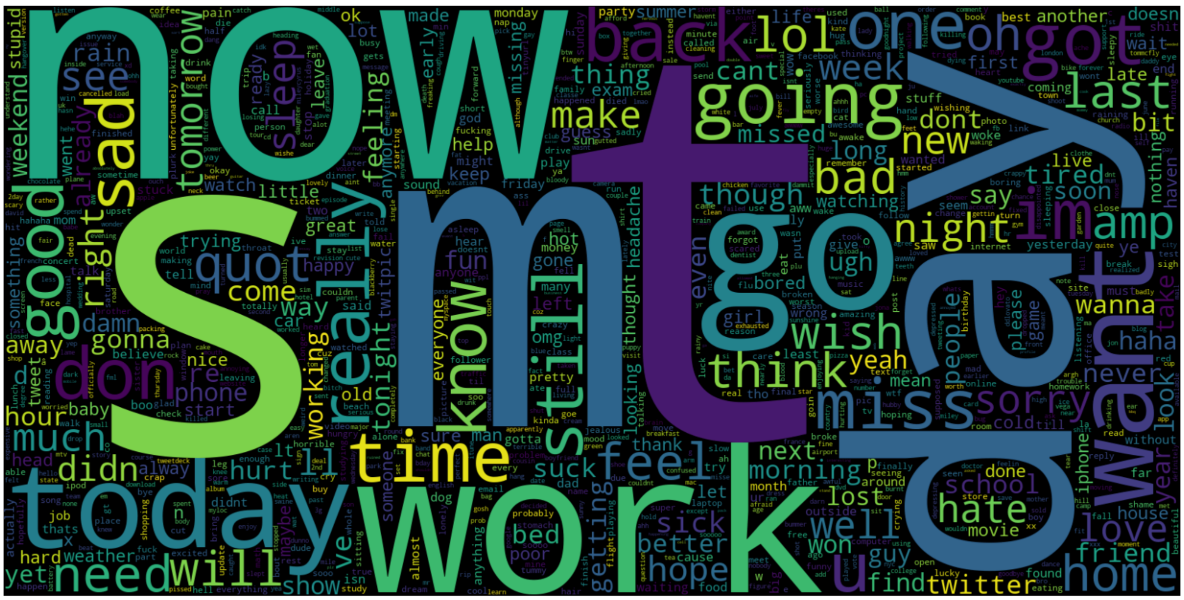
#### Word-Cloud for Negative tweets

In [13]:
```
# Assuming the previous steps have been completed and 'processed_text' is defi

# Extract the first 800,000 processed texts for negative data (or any required
data_neg = processed_text[:800000]  # Note: processed_text was defined previou

# Create a WordCloud for the negative text data
plt.figure(figsize=(20, 20))
wc = WordCloud(max_words=1000, width=1600, height=800, collocations=False).gen

# Display the WordCloud
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')  # Turn off axis labels for better visualization
plt.show()
```



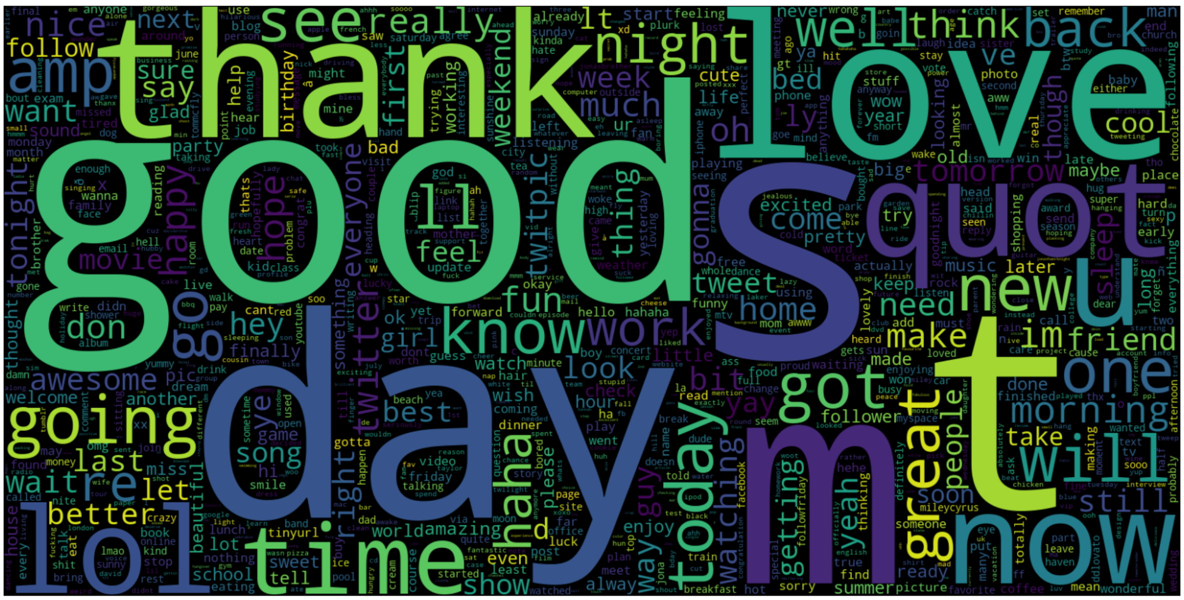

**Word-Cloud for Positive tweets**

In [15]:
```
# Assuming 'processed_text' was defined after the preprocessing step

# Extract the positive sentiment data (assuming it's from the 800,000th entry o
data_pos = processed_text[800000:]  # Correct variable name

# Create a WordCloud for the positive text data
wc = WordCloud(max_words=1000, width=1600, height=800, collocations=False).gene

# Display the WordCloud
plt.figure(figsize=(20, 20))
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')  # Turn off the axis for better visualization
plt.show()
```




## Splitting the data

In [17]:
```
from sklearn.model_selection import train_test_split
```

In [18]:
```
from sklearn.model_selection import train_test_split

# Ensure that 'processed_text' and 'dataset["sentiment"]' are defined
# processed_text contains preprocessed tweet text
# sentiment contains the target variable (labels)

X_train, X_test, y_train, y_test = train_test_split(processed_text, dataset['se
                                                    test_size=0.05, random_sta

print('Data Split done.')
```

Data Split done.

In [19]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [20]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Create and fit the TfidfVectorizer
vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=50000)
vectorizer.fit(X_train)

print('Vectorizer fitted')
# Use get_feature_names_out() instead of get_feature_names()
print('No. of feature_words: ', len(vectorizer.get_feature_names_out()))
```

```
Vectorizer fitted
No. of feature_words:  50000
```

In [21]:
```
X_train = vectorizer.transform(X_train)
X_test = vectorizer.transform(X_test)
print(f'Data Transformed')
```

```
Data Transformed
```

# Creating and Evaluating Models

Creating 3 different types of model of our sentimental analysis probelms.

- **Bernoulli Naive Baye(Bernoulli)**
- **Linear Support Vector Classificatio (LinearSVC)**
- **Logistic Regression (LR)**

In [23]:
```
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
```

## Evaluation Model Function

In [25]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [26]:
```
# Importing necessary libraries
import re
import pickle
import numpy as np
import pandas as pd

import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Setting a more appealing style for plots
sns.set(style='whitegrid')
plt.style.use('ggplot')  # Use ggplot style for better visual appeal
```

In [27]:
```
def model_evaluate(model):
    y_pred = model.predict(X_test)

    # classification report
    print(classification_report(y_test, y_pred))

    # confusion report
    cf_matrix = confusion_matrix(y_test, y_pred)

    categories = ['Negative', 'Positive']

    group_names = ['True Neg', 'False Pos','False Neg','True Pos']

    group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatte

    labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)

    sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues',fmt = '',
                xticklabels = categories, yticklabels = categories)

    plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)
    plt.ylabel("Actual values"   , fontdict = {'size':14}, labelpad = 10)
    plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

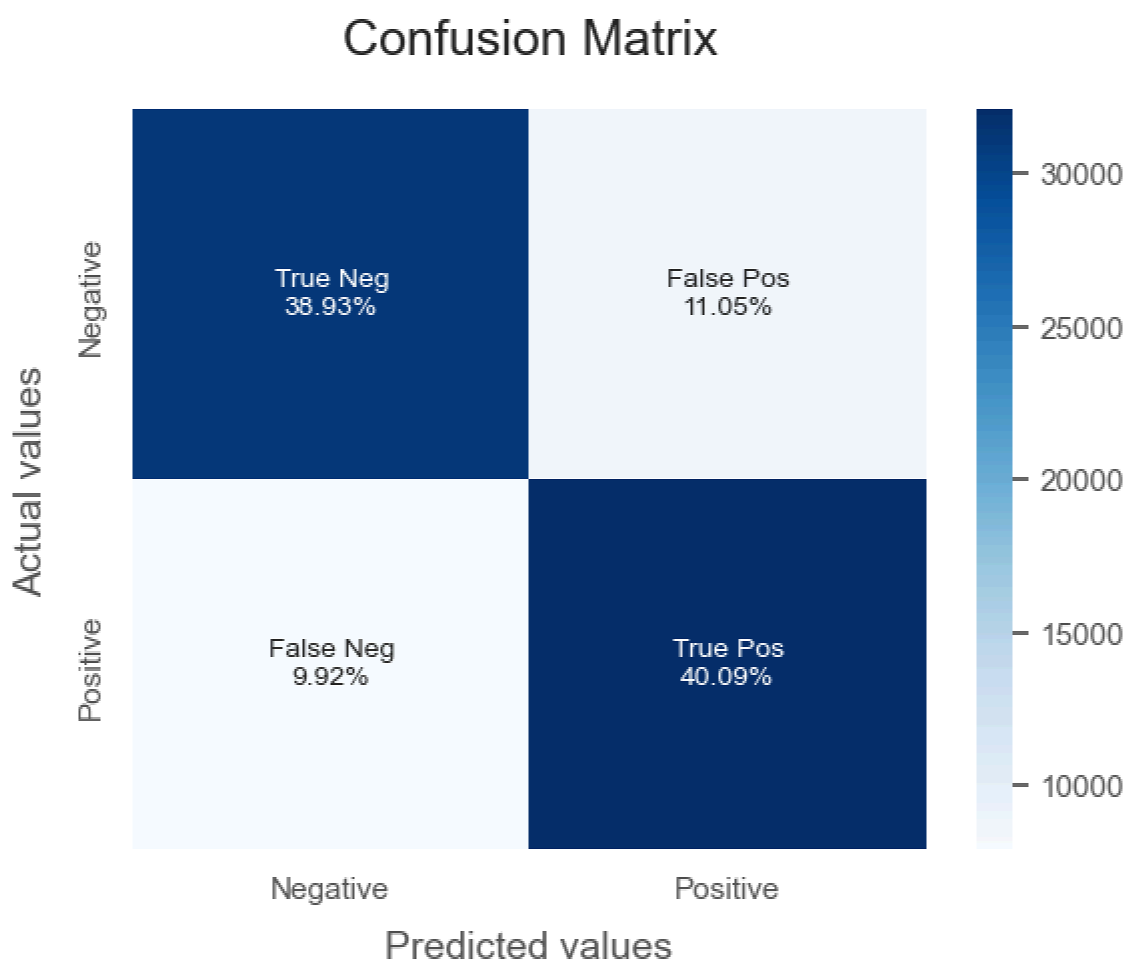
# BernoulliNB Model

In [29]:
```
BNBmodel = BernoulliNB(alpha = 2)
BNBmodel.fit(X_train, y_train)
model_evaluate(BNBmodel)
```

```
              precision    recall  f1-score   support

           0       0.80      0.78      0.79     39986
           1       0.78      0.80      0.79     40014

    accuracy                           0.79     80000
   macro avg       0.79      0.79      0.79     80000
weighted avg       0.79      0.79      0.79     80000
```



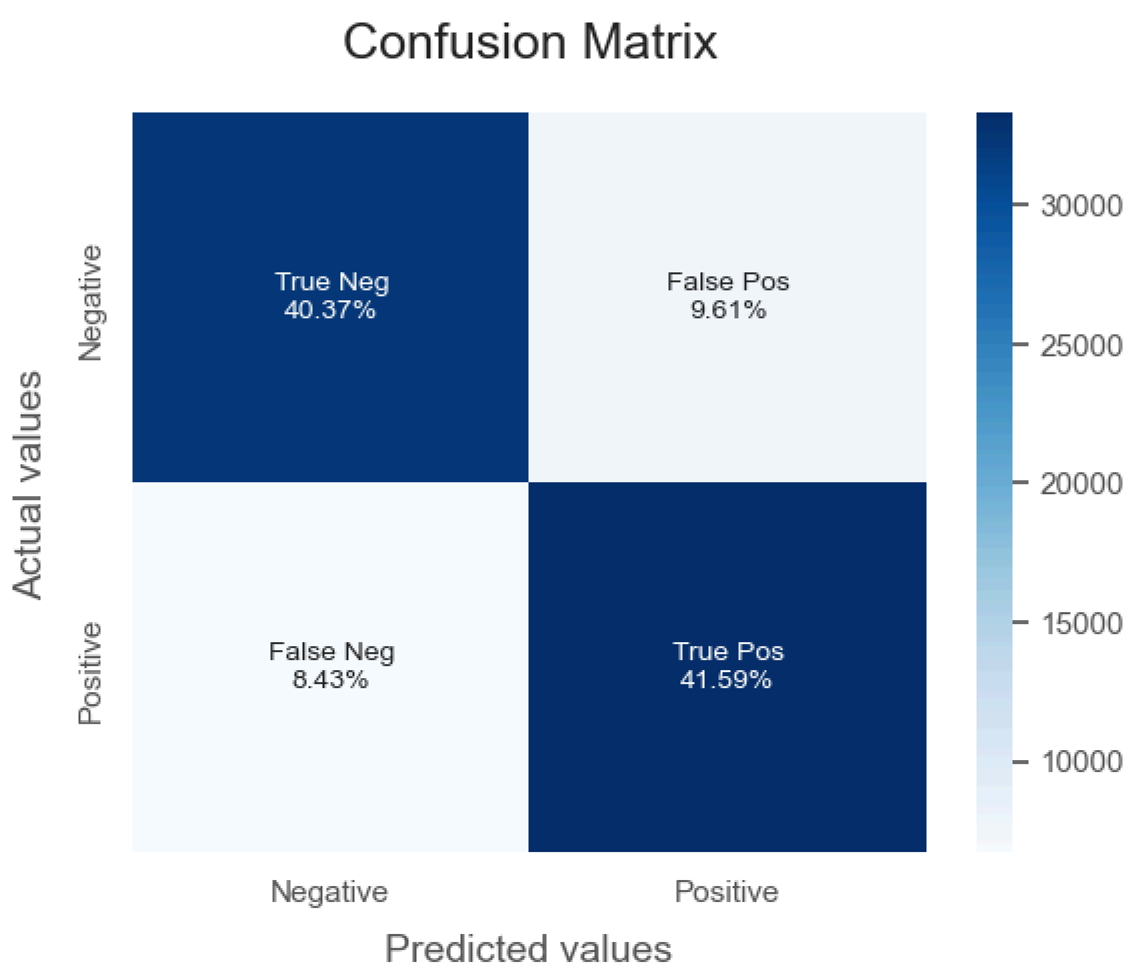
# LinearSVC Model

In [31]:
```
SVCmodel = LinearSVC()
SVCmodel.fit(X_train, y_train)
model_evaluate(SVCmodel)
```

```
C:\Users\mohan\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:31: Future
Warning: The default value of `dual` will change from `True` to `'auto'` in
1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
              precision    recall  f1-score   support

           0       0.83      0.81      0.82     39986
           1       0.81      0.83      0.82     40014

    accuracy                           0.82     80000
   macro avg       0.82      0.82      0.82     80000
weighted avg       0.82      0.82      0.82     80000
```



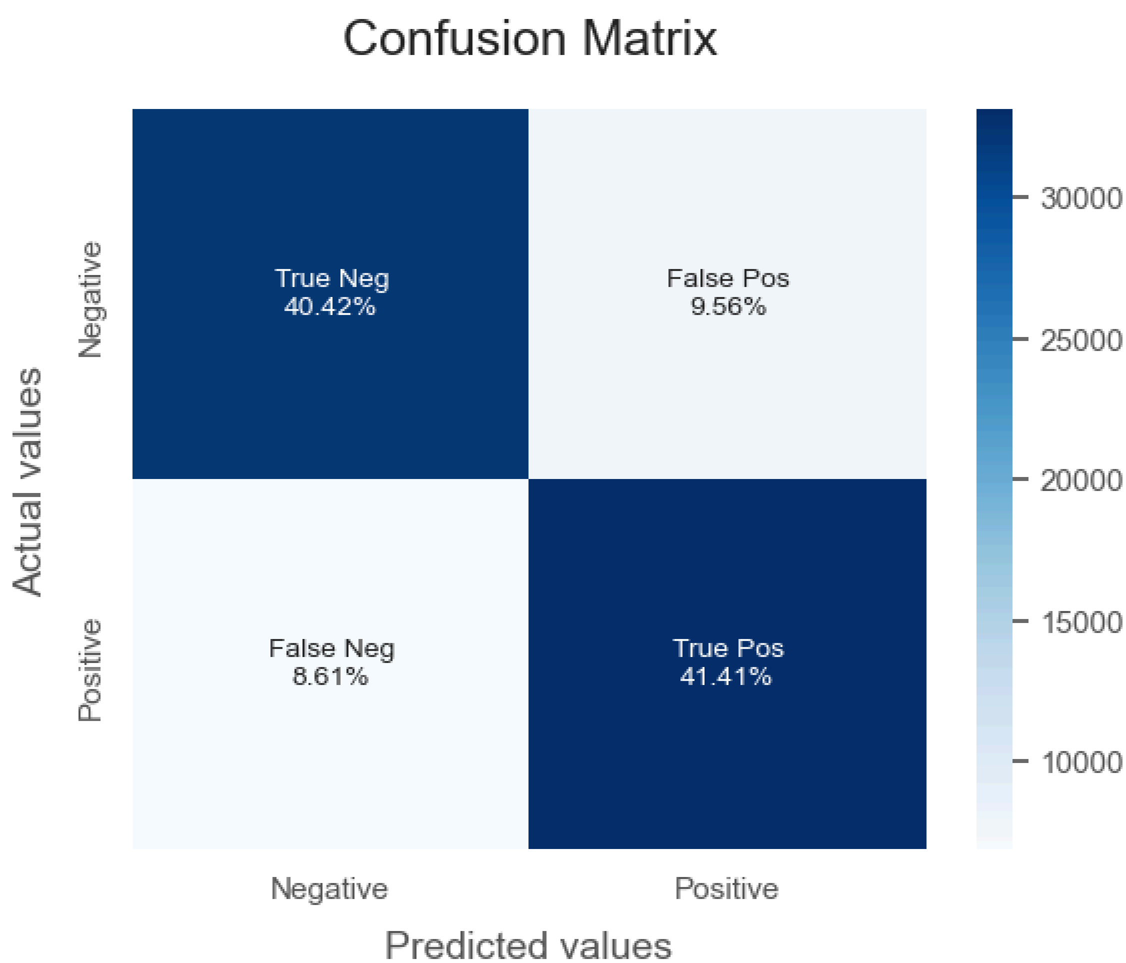
# Logistic Regression Model

In [33]:
```
LRmodel = LogisticRegression(C =2, max_iter=1000, n_jobs=1)
LRmodel.fit(X_train, y_train)
model_evaluate(LRmodel)
```

```
              precision    recall  f1-score   support

           0       0.82      0.81      0.82     39986
           1       0.81      0.83      0.82     40014

    accuracy                           0.82     80000
   macro avg       0.82      0.82      0.82     80000
weighted avg       0.82      0.82      0.82     80000
```




We can clearly see that the **Logistic Regression Model** performs the best out of all the different models that we tried. It achieves nearly 82% accuracy while classifying the sentiment of a tweet.

Although it should also be noted that the BernoulliNB Model is the fastest to train and predict on. It also achieves 80% accuracy while calssifying.

# Saving the model

In [36]:
```
file = open('vectoriser-ngram-(1,2).pickle','wb')
pickle.dump(vectorizer, file)
file.close()

file = open('Sentiment-LR.pickle','wb')
pickle.dump(LRmodel, file)
file.close()

file = open('Sentiment-BNB.pickle','wb')
pickle.dump(BNBmodel, file)
file.close()
```

In [37]:
```
def load_models():
    '''
    Replace '..path/' by the path of the saved models.
    '''

    # Load the vectoriser.
    file = open('..path/vectoriser-ngram-(1,2).pickle', 'rb')
    vectoriser = pickle.load(file)
    file.close()
    # Load the LR Model.
    file = open('..path/Sentiment-LRv1.pickle', 'rb')
    LRmodel = pickle.load(file)
    file.close()

    return vectoriser, LRmodel

def predict(vectoriser, model, text):
    # Predict the sentiment
    textdata = vectorizer.transform(preprocess(text))
    sentiment = model.predict(textdata)

    # Make a list of text with sentiment.
    data = []
    for text, pred in zip(text, sentiment):
        data.append((text,pred))

    # Convert the list into a Pandas DataFrame.
    df = pd.DataFrame(data, columns = ['text','sentiment'])
    df = df.replace([0,1], ["Negative","Positive"])
    return df

if __name__=="__main__":
    # Loading the models.
    #vectoriser, LRmodel = load_models()

    # Text to classify should be in a list.
    text = ["I hate our president",
            "I Love you.",
            "Yes! We can win"]

    df = predict(vectorizer, LRmodel, text)
    print(df.head())
```

```
                   text sentiment
0  I hate our president  Negative
1           I Love you.  Positive
2       Yes! We can win  Positive
```

In [ ]: